**CSC 4420 Midterm Study Guide**

**Winter 2025**

**Chapter 1: Introduction**

1. Understand what an Operating System is

* Operating systems can be viewed from two viewpoints: resource managers and extended machines. In the resource-manager view, the operating system’s job is to manage the different parts of the system efficiently. In the extended-machine view, the system provides the users with abstractions that are more convenient to use than the actual machine.
* The abstractions include processes, address spaces, and files. Operating systems handles requests, prevents errors and improper use, serves as a file system and operates a windowing system where the graphical user interface (GUI) presents data in a window format to the user interface (UI).

1. Retain knowledge on history of Operating Systems

* Operating systems have a long history, starting from the days when they replaced the operator, to modern multiprogramming systems. Highlights include early batch systems, multiprogramming systems, and personal computer systems.
* The early operating systems were focused on managing I/O operations and scheduling jobs in the batch processing systems. The modern operating systems are focusing on efficient resource management, providing a user-friendly interface, robust security features, seamless integration with cloud services, support for multiple processors and diverse hardware, and handling complex applications like multimedia while maintaining high performance across a variety of devices, including mobile and cloud-based environments.
* Modern operating systems typically include built-in networking capabilities, allowing users to connect to the internet, access shared network drives, and send data to other devices on the network.

1. Know the key Computer Hardware components

* Operating system interacts with several hardware components including the CPU (Central Processing Unit), RAM (Random Access Memory), storage devices (hard drives), input/output devices (keyboard, mouse), and the motherboard which connects all these components together.
* Essentially, the OS manages how these hardware parts work together to run applications and perform tasks on a computer. Memory is one of key hardware components and structured as a hierarchy based on the speed, cost and capacity. Main memory is the primary internal workspace that stores data temporarily with limited capacity and virtual memory is a memory management technique that uses a computer’s secondary memory to make up for a lack of physical memory.

1. Know the fundamental Operating System types and structures

* Operating systems can be structured as monolithic, layered, microkernel/client-server, virtual machine, or exokernel/unikernel systems. Regardless, the basic concepts on which they are built are processes, memory management, I/O management, the file system, and security. The main interface of an operating system is the set of system calls that it can handle. These tell us what it really does.

1. Know what system Calls are

* System call is a mechanism that allows a user program to request a service from the operating system kernel, acting as a bridge between the user space (applications) and the kernel space, enabling actions like opening files, reading data, creating processes, or managing network connections, all by essentially "calling" a specific function provided by the kernel to perform the desired operation. Essentially, it's the primary way a program interacts with the operating system.
* It is very commonly to make a system call to trap the kernel. The system calls can act as a controlled way to switch to kernel mode from user mode. While interrupts are asynchronous events trigged by external hardware.

1. fundamental C

* C is used as the primary language to build the Linux kernel and many of its system-level components, making it a foundational element of the operating system, allowing for high performance and direct access to hardware capabilities; essentially, understanding basic C is crucial for working deeply with Linux systems.
* In Linux, everything is a file. This includes hardware devices, processes, directories, regular files, sockets, links, and so on. Also, the file system is usually divided into data blocks and inodes. With that said, you can think of inodes as the foundation of the Linux file system.
* A pipe in Linux/Unix is a mechanism that allows the output of one command to be used as the input for another command, essentially connecting the standard output of a process to the standard input of another, enabling users to chain multiple commands together to efficiently filter and manipulate data on the command line. When using pipe in the script, a good practice is to close one end of pipe if using another end. The same concept can be applied to the file descriptors.

**Chapter 2:**

1. Understand what a process is, how a process is created and how the state transitions work.

* Process is a separate instance of a running program with its own memory space, meaning any time you execute a command or application, it becomes a process with its own unique identifier (PID) that allows the operating system to track and manage its execution; essentially, it's a program actively running on the system.
* A new process can be created inside a process by making a fork system call. Processes are organized in a tree-like structure where a parent process can create child processes, allowing for efficient management of related tasks. A process can be in different states like running, ready and blocked. One state can transit to another state when the condition is met.

1. Know what a thread is, how a thread is created, how a thread is managed, what the relationships are between processes and threads and what different thread models are.

* Threads are the smallest unit of execution within a process. They share the same memory space as the parent process. Threads can execute independently, but they are not completely independent like separate processes.
* Creating a new thread doesn’t need to allocate separate memory space so the creation is light weighted. Context switching between threads is faster because threads share the same memory space. Communications are faster between threads.
* Each thread has its own thread table which can be stored in either user space or kernel space. There are some pros and cons on where the thread table is stored.
* Different thread model is suitable for different applications, like event-driven servers is designed to respond to events, meaning it waits for specific occurrences (like a user action, data change, or system notification) and then executes code only when those events happen, instead of constantly polling for updates; this allows for efficient resource usage and highly responsive systems, often utilizing messaging brokers like manage event streams.

1. Know what synchronization and Inter-process Communication are

* Deadlock is a situation where two or more processes are stuck waiting for each other to release resources held by another process in a circular dependency where none of the processes can continue execution, effectively halting the system. Many ways can prevent the deadlock like, mutual exclusion, priority ordering, timeout mechanism, etc. Deadlock is a specific type of race condition where the outcome is always a complete standstill.
* Race Condition occurs when two or more threads or processes try to access and modify a shared resource (critical region) simultaneously, leading to unpredictable results depending on the order of execution. Various synchronization mechanisms like locks(mutexes), Peterson’s solution, TSL/XCHG, semaphores, monitors and message passing can be used to avoid or minimize the race condition.
* Processes can synchronize with one another using synchronization and interprocess communication primitives, for example, semaphores, monitors, or messages. These primitives are used to ensure that no two processes are ever in their critical regions at the same time, a situation that leads to chaos. A process can be in one of running, ready, and blocked states. The state can be changed when the process executes one of interprocess communication primitives.

1. Know pros and cons on common scheduling algorithms

* A great many scheduling algorithms have been studied. Some of these are primarily used for batch systems, such as shortest-job-first scheduling. Others are common in both batch systems and interactive systems. These algorithms include round robin, priority scheduling, multilevel queues, guaranteed scheduling, lottery scheduling, and fair-share scheduling. Some systems make a clean separation between the scheduling mechanism and the scheduling policy, which allows users to have control of the scheduling algorithm.
* In interactive systems, one of the goals is to minimize turnaround time which refers to the process of prioritizing which tasks to run on the CPU based on the total time it takes for a process to complete from the moment it is submitted to the system until it finishes execution, including any waiting time in the queue
* Two types of scheduling: preemptive scheduling means that a currently running process can be interrupted at any time by the operating system to allow a higher priority process to execute, while non-preemptive scheduling means a process will run uninterrupted until it finishes or voluntarily yields control to another process, even if a higher priority process is waiting to run.